(NeXT Tip #49) Symbols as Array Indexes

Christopher Lane (*lane[at]CAMIS.Stanford.EDU*)
Fri, 25 Feb 1994 14:30:22 -0800 (PST)

(Another of those annoying C programming style tips.)

Looking through application source code on /LocalDeveloper/Examples (which we
import from outside -- all around the Internet), I noticed a high incidence of
the use of numbers, rather than symbols, as array indexes.

Some examples:

```
textBuf = &cursor[10];
sscanf(&s[7], " %d", &i);
mx[7] = -sin(theta) * cos(phi);
static int scrambledSize3[NUMOFCHALLENGES][9];
if(s[13] == elinfo[j].name[0] && elinfo[j].name[1] == ' ') break;
char host[MAX_VPATH], location[MAX_VPATH], filename[MAX_VPATH], mode[11];
while((dst[-1] == '\n') && (dst[-2] == '>') && (dst[-3] == '\n')) dst -= 2;
```

(The last example is particularly interesting and only make sense when you
consider that arrays and pointers are interchangeable in 'C'.) Although
's[13]' may have had an obvious meaning at the time, as you and others look at
the code later, this can make it difficult to decipher and fix problems.

I did a simple search to measure how often numbers were used to index (or
size) arrays in the *.m (but not *.h) files in the Examples directory -- I've
only included counts for more than 1 occurrence and a '*' ~= ten occurrences:

```
[-1]   12 *
 [0] 1264 ************************************************************ ... ***
 [1]  603 *************************************************************
 [2]  496 **************************************************
 [3]  199 *******************
 [4]   91 **********
 [5]   96 **********
 [6]   55 ******
 [7]   34 ***
 [8]   28 ***
 [9]   19 **
[10]   39 ****
[11]    8 *
[12]   10 *
[13]   10 *
[14]   10 *
[15]   15 **
[16]   24 **
```

The curve seems reasonable; you'd expect lots of 'x[0]' and 'y[1]' indexes and
'a = z[3]' is used often to define (non-NeXT) bounding regions. (The curve
takes longer to trail off to zero than I would have guessed.) There are some
(not too surprising) minor spikes at '[10]' and '[16]'.

How can you avoid using numbers as indexes into an array of heterogeneous
elements? One traditional way is to #define symbolic indexes:

#define iJulianDate 0
#define iUniversalTime 1
...
#define iNextNewMoon 14
#define iNextLunation 15

Another way to do this is with enumeration:

typedef enum {NRG_NMOD, NRG_HBOND, NRG_NONB, NRG_ELST} MTypes;

Then use 'x = array[NRG_NONB]'. Particularly if you don't care about order
and you want to be able to easily, arbitrarily add elements. (You can anchor
enum elements to numbers if needed by doing ', NRG_HBOND=4,' etc.) Of course,

it's easy to concoct symbols that are less mnemonic than small integers!

Another enum example is one I've described earlier regarding argc & argv:

typedef enum {PROGRAM, MODE, FILENAME, SEGMENT, SECTION, ARGC} ARGUMENTS;

This is only useful for fixed position arguments (no options) and allows you to do a test like '(argc == ARGC)' to see if all the arguments have been supplied and access 'argv[SEGMENT]' rather than the traditional 'argv[3]'.

One place you can avoid constants in array sizing is:

char *types[4] = {"graph", "palette", "bundle", NULL};

Where you walk the array checking for set inclusion and use NULL to signal the last element. The compiler is reasonably cooperative and lets you simply do:

char *types[] = {"graph", "palette", "bundle", NULL};

Which is easier to update without introducing inconsistancy.

The symbolic index issue is a special case of my personal programming rule of thumb that numbers greater than +/- 2 in the body of a program are suspect and should be rethought and #define'd or eliminated in some other fashion.

- Christopher 'ColorScroller.m:#define M(x) (170.-.3*(170-x.))/256.'